

# Syntactic and semantic classification of verb arguments using dependency based and rich semantic features

Francesco Elia\*  
Università di Roma “La Sapienza”

## Abstract

Corpus Pattern Analysis (CPA) has been the topic of Semeval 2015 Task 15, aimed at producing a system that can aid lexicographers in their efforts to build a dictionary of meanings for English verbs using the CPA annotation process. CPA parsing is one of the subtasks which this annotation process is made of and it is the focus of this report. A supervised machine-learning approach has been implemented, in which syntactic features derived from parse trees and semantic features derived from WordNet and word embeddings are used. It is shown that this approach performs well, even with the data sparsity issues that characterize the dataset, and can obtain better results than other system by a margin of about 4% f-score.

## 1 Introduction

Recent research on Corpus Pattern Analysis, a corpus-driven technique for identifying and assigning meaning to patterns of word usage in text, suggests that it may be useful to build a semantic resource that can be used in several NLP applications. As of now, the main output of CPA is the Pattern Dictionary of English Verbs (<http://www.pdev.org.uk>), a manually built collection of patterns with entries for each verb in the English language. Task 15 at SemEval 2015 focused on Corpus Pattern Analysis and PDEV, with the aim of producing systems that can automatically build a pattern dictionary for verbs using CPA. To perform this, several stages of processing are needed. The first one, called “CPA parsing”, is the focus of this report. The CPA parsing task requires a system to identify and classify, from a syntactic and semantic perspective, the relevant arguments for a target verb in a given sentence. The task is similar to Semantic Role Labeling, but single tokens are identified in the dependency parsing paradigm, rather than phrases in the constituency parse tree. In this

---

\*1243034.elia@studenti.uniroma1.it

report, a system that can perform this task using a learning-based approach is illustrated, by training three maximum entropy classifiers that perform argument identification, syntactic and semantic classification. A rich set of both syntactic and semantic features has been used, showing that they are effective at performing the given task. The system improves on previous results on the same task, with an f-score increase of almost 4% on the best performing system. These results show that, despite the data sparsity that characterizes the dataset provided for this task, a learning-based approach can perform well with the use of descriptive features and that, most likely, this approach would perform even better if more data was available.

## 2 CPA parsing

CPA parsing requires a system to analyze a sentence, extract the verb’s main arguments and tag them both syntactically (see Table 2 for the syntactic tagset) and semantically (using the CPA ontology<sup>1</sup>). As an example, let’s consider the following sentence, where the verb “continue” is the target verb, that is the verb whose arguments have to be extracted:

European politicians continue to plead, sincerely, that Yugoslavia should endure.

The goal of CPA parsing is to annotate this sentence as shown in Table 1.

Token	Syntactic tag	Semantic tag
European		
politicians	subj	Human
continue	v	-
to	advprep	LexicalItem
plead	acomp	Activity
,		
sincerely		
,		
that		
Yugoslavia		
should		
endure		
.		

Table 1: An example of correctly annotated sentence using CPA syntactic and semantic tag sets

In this task, the target verb is the only information passed to the system, since it is already marked in each sentence in the dataset and thus does not

<sup>1</sup>Available at: <http://pdev.org.uk/#onto>

need to be identified. The most important part in solving this task is undoubtedly identifying which tokens in the sentence are actual arguments for the verb, because mistakes at this stage will reflect on all the following steps and will negatively impact the performance of the system. For this reason, similarly to the approach used in [1], I divided the CPA parsing task into three smaller subtasks: argument identification, syntactic classification and semantic classification. Although a single classifier could be used to jointly identify and syntactically classify each token (i.e., by directly assigning a label of “subj”, “obj”, etc... when a token is an argument and assigning a label of “none” when the token is not) this division allows to separately study which features work best for argument identification and thus improve it without having to worry about the syntactic classification yet.

Tag	Definition
obj	Object
subj	Subject
advprep	Adverbial preposition or other Adverbial/Verbal link
acomp	Adverbial or Verb complement
scomp	Noun or Adjective complement
iobj	Indirect object

Table 2: Syntactic tagset for CPA parsing

The three steps of argument identification, syntactic and semantic classification all use a machine-learning approach, implemented with a maximum entropy classifier trained using the Stanford Classifier implementation<sup>2</sup>. Maximum entropy models, also known as log-linear or exponential, have been widely used for several NLP tasks, for example question classification, named entity recognition and sentiment analysis; they provide the capability to be trained with several thousands features, which is very common for applications in NLP, and, to a certain extent, they can distinguish between more and less relevant features by automatically assigning higher weights to the former [2]. Figure 1 shows the data processing pipeline and how these three modules are arranged with respect to the other components of the system, namely preprocessing and feature extraction.

The preprocessing step, described in detail in Section 2.1, is needed to parse the input sentence and compute syntactic and semantic information that is later used for feature extraction. Several features and their combinations have been tested for each classifier and the best performing ones have been chosen using a greedy hill-climbing algorithm as described in Section 2.2. The argument identification, syntactic and semantic classification processes are described in detail in Sections 2.3, 2.4 and 2.5 respectively.

<sup>2</sup> Available at: <http://nlp.stanford.edu/software/classifier.shtml>

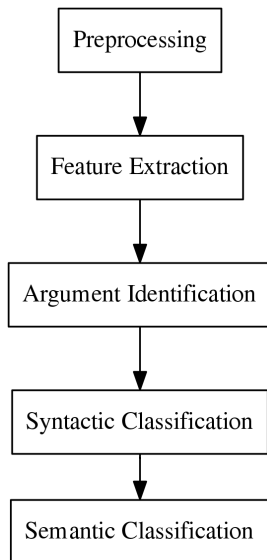


Figure 1: Data processing pipeline

## 2.1 Preprocessing

The preprocessing step augments each sentence in the dataset with the linguistic and syntactic information needed to later perform the feature extraction step. All the syntactic information is obtained by using the Stanford CoreNLP pipeline [3]: in particular, each sentence in the dataset is augmented with POS tags, lemmas, a constituency and a dependency parse tree.

## 2.2 Features and feature selection

Despite the system has to work on three different tasks, features have not been created specifically for each one of them: considering the fact that the argument identification and syntactic classification tasks may require very similar features and that these features, despite syntactic in nature, may also benefit the semantic classification task anyway, a unique set of features has been devised and the work of selecting the best performing subset of features for each task is left to the feature selection algorithm. The algorithm starts with an initial, possibly empty, subset of features (provided by the user) and tries one new feature at a time: a new classifier is trained and tested with the new set of features and the new feature is kept if it improves the f-score on the dev set. When there are no more features to be added the algorithm terminates.

**Data:**  $S$  = initial subset of features,  $A$  = all the features excluding those in  $S$

**Result:** Subset of features  $S$  with the highest f-score

bestFscore = 0;

**while**  $A$  is not empty **do**

    feature = sample( $A$ );

$A$ .remove(feature);

    fscore = fscore( $S \cup \{feature\}$ );

**if** fscore > bestFscore **then**

$S$ .add(feature);

        bestFscore = fscore;

**end**

**end**

return  $S$ ;

**Algorithm 1:** Hill-climbing search for best combination of features

For compactness reasons, the complete list of features, along with a description for each one, is provided in Appendix A.

## 2.3 Argument identification

Argument identification is performed by a binary classifier that works locally on each token in the sentence and decides whether it is an argument or not. The training set for this classifier is obtained by taking all the tokens in the training set and labeling them with the "argument" class when they have a syntactic label or with "none" otherwise. As an example, how a sample sentence gets annotated by the argument identification classifier:

Token	Class
Universities	argument
continued	verb
to	argument
languish	argument
through	
the	
eighties	

Table 3: Output of the argument identification classifier on the example sentence. Tokens that do not represent arguments are labeled with "none" but, for simplicity, it is not shown in the table.

It must be stressed that argument identification is probably the most important step in the whole process, because if an argument is missed or a non-argument is erroneously identified as one, these errors will necessarily propagate through the pipeline rendering the execution of the following steps meaningless.

## 2.4 Syntactic classification

This classifier operates on the output of the argument identification step. Tokens identified as arguments are passed to the syntactic classifier, which assigns each of them to one of the 6 possible syntactic classes (subj, obj, iobj, advprep, acomp, scomp). Continuing with the same example sentence, three tokens have been identified as arguments and they have to be classified syntactically:

Token	Syntactic class
Universities	subj
continued	verb
to	advprep
languish	acomp
through	
the	
eighties	

Table 4: Output of the syntactic classifier on the example sentence.

## 2.5 Semantic classification

The semantic classifier operates on the output of the previous steps and works in a similar fashion. Each token that has been recognized as an argument gets assigned a semantic label, chosen from the 118 labels present in the training set. Here's how the example sentence that has been considered up to now gets annotated:

Token	Syntactic class	Semantic class
Universities	subj	<b>Institution</b>
continued	verb	-
to	advprep	<b>LexicalItem</b>
languish	acomp	<b>Action</b>
through		
the		
eighties		

Table 5: Output of the semantic classifier on the example sentence, which is now fully annotated.

## 3 Experiments

### 3.1 Overview of the dataset

The performance of the system has been evaluated on the SemEval 2015 Task 15 dataset. The training set is made of 3249 sentences for 21 different verbs, while the test set contains 1280 sentences for 7 verbs. It must be noted that the verbs contained in the training set are different than those contained in the test set, in order to encourage researchers to work on systems that could generalize to different verbs than those they have been trained on. Table 6 shows the number of sentences for each verb in the training and test set.

Verb	# of sentences	Verb	# of sentences
allow	150	sabotage	77
crave	75	recall	263
launch*	207	claim	212
propose	169	applaud	198
execute*	213	veto	123
pray*	180	plan	130
announce	228	account	155
battle*	190	<u>undertake</u>	228
plead	205	<u>crush</u>	170
abandon	172	<u>operate</u>	140
answer	171	<u>apprehend</u>	123
abort	60	<u>appreciate</u>	215
squeal	20	<u>continue</u>	203
disable	51	<u>decline</u>	201

Table 6: Number of sentences for each verb in the dataset. Underlined verbs are part of the test set. Verbs marked with \* have been chosen for the dev set.

The training set has a total of 7122 tokens with a syntactic and semantic label. Figure 3.1 shows how syntactic classes are distributed in the training set: the most frequent class (rank 0) is *subj* followed by *obj*, *acomp*, *advprep*, *scomp* and finally *iobj*. The last two classes (*iobj* and *scomp*) are an order of magnitude less frequent than all the other ones, but considering the fact that they do not appear in the test set, and thus do not affect the performance of the system, the frequency of the other classes is pretty balanced.

Unfortunately, as can be seen from Figure 3, the distribution of semantic classes is instead very skewed, with the first three most common classes (i.e., Human, LexicalItem, Action) accounting for 57% of the total number of examples. Out of the 118 semantic classes that appear in the training set, 77 of them do so less than 10 times and this is potentially an issue for the semantic classification task, that will be shown to be the less performing part of the system.

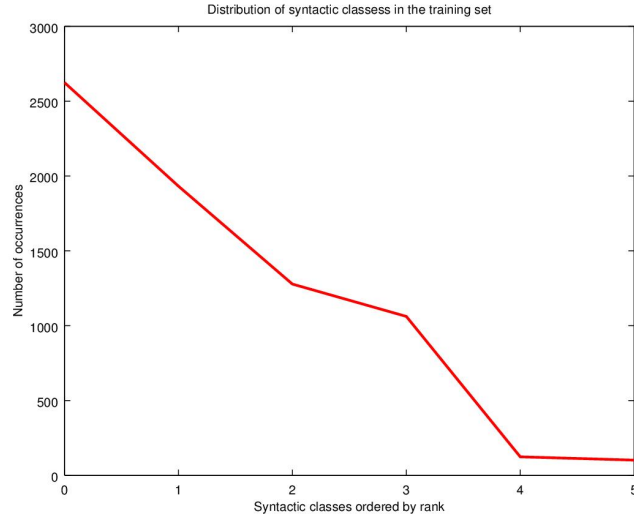


Figure 2: Distribution of syntactic classes in the training set.

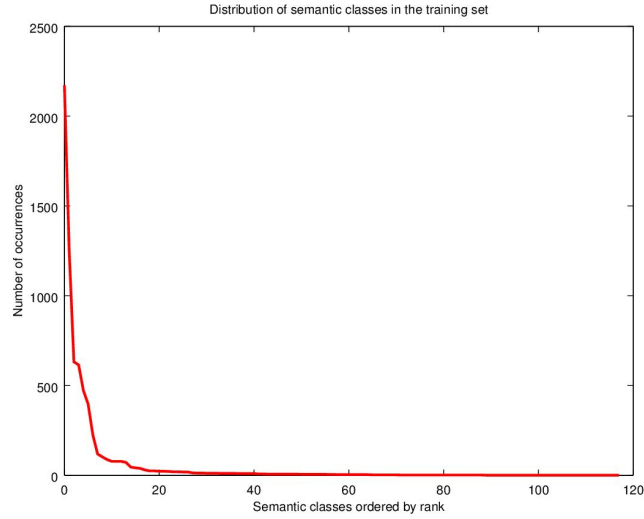


Figure 3: Distribution of semantic classes in the training set. As can be seen, the frequency of a class is roughly inversely proportional to its rank, leading to a very skewed distribution where the top 3 most frequent classes constitute more than half of the training set.

## 3.2 Metrics

The metric used for the evaluation is the average f-score of the system over each verb in the test set:

$$F1_{verb} = \frac{2 \cdot \text{Precision}_{verb} \cdot \text{Recall}_{verb}}{\text{Precision}_{verb} + \text{Recall}_{verb}}$$



$$\text{Score} = \frac{\sum_{verb \in V} \text{F1}_{verb}}{|V|}$$

Where  $V$  is the set of verbs contained in the test set. Precision and recall are calculated as follows:

$$\text{Precision} = \frac{\text{Correct tags}}{\text{Retrieved tags}}$$

$$\text{Recall} = \frac{\text{Correct tags}}{\text{Reference tags}}$$

A tag is considered correct when it is placed on the exact same token and matches the annotation in the gold standard. For this reason, a good performance of the argument identification module is essential for good final results, seeing as errors at this stage will be reflected on all the following ones: if arguments are not identified this will impact recall for syntactic and semantic classification and if tokens are erroneously identified as arguments this will impact their precision, no matter what syntactic/semantic class is assigned to those tokens.

### 3.3 Results of feature selection

Applying the feature selection algorithm results in three subsets of features, one for each of the three classifiers used in the pipeline, as shown in the following table.

Feature name	ArgId	SynClass	SemClass
TOKENLEMMA	•	•	•
TOKENWORD			
TOKENPOS	•		•
LEMMASAROUNDTOKEN			•
WORDSAROUNDTOKEN			
POSAROUNDTOKEN			
TOKENISVERB			
TOKENISPREPOSITIONOFVERB	•		
TOKENPHRASETYPE	•		
TOKENPHRASESTRUCTURE		•	
TOKENISSUBJOROBJ		•	
TOKENISVERBCHILD	•	•	
TOKENISCAPITALIZED		•	•
TOKENCONTAINSDIGIT			•
TOKENISUPPERCASE			•
TOKENRELFROMVERB	•	•	
TOKENISUNIQUESUBJOROBJ			
VERBLEMMA			

Feature name	ArgId	SynClass	SemClass
VERBPOS		•	
LEMMASAROUNDVERB		•	
POSAROUNDVERB		•	
VERBVOICE		•	
VERBPOSITION		•	
ISVERBPREPOSITIONAL			
VERBBY			
VERBPHRASESTRUCTURE		•	
VERBISROOT		•	
VERBHASNsubj	•	•	
VERBHASNsubjPASS	•	•	
VERBHASDOBJ	•	•	
VERBHASIOBJ			
VERBHASCcomp		•	
VERBHASAcOMP			
VERBHASXCOMP	•		
VERBPARENTLEMMA		•	•
VERBPARENTPOS			
VERBFIRSTVpPARENTLEMMA	•		
RELVERBPARENTTOVERB	•		
TOKENDIRDPATHFROMVERB	•	•	•
TOKENDIRDPATHFROMVERBWITHLEMMA		•	•
TOKENDIRDPATHFROMVERBWITHPOS	•		
TOKENUNDDPATHFROMVERB	•	•	
TOKENUNDDPATHFROMVERBWITHLEMMA			•
TOKENUNDDPATHFROMVERBWITHPOS	•	•	
DIRDPATHVERBVpPARENTTOVERB	•		
DIRDPATHVERBVpPARENTTOVERBWITHLEMMA	•	•	
DIRDPATHVERBVpPARENTTOVERBWITHPOS	•	•	
TOKENDIRDPATHFROMVERBVpPARENT	•	•	
TOKENDIRDPATHFROMVERBVpPARENTWITHLEMMA	•	•	
TOKENDIRDPATHFROMVERBVpPARENTWITHPOS		•	
TOKENPARENTDIRDPATHFROMVERB	•	•	
TOKENPARENTDIRDPATHFROMVERBWITHLEMMA	•		
TOKENPARENTDIRDPATHFROMVERBWITHPOS	•	•	
TOKENDIRDPATHFROMVERBPARENT	•		
TOKENDIRDPATHFROMVERBPARENTWITHLEMMA			
TOKENDIRDPATHFROMVERBPARENTWITHPOS		•	
TOKENCPATHFROMVERB	•	•	
TOKENCPATHFROMVERBPARENT	•	•	
VERBHYPERNYMSMcs	•		•
HYPERNYMSMcs			•
HYPERNYMSDISAMBIGUATED			
TOKENSIMILARWORDS			•

Feature name	ArgId	SynClass	SemClass
VERBSIMILARWORDS			•
TOKENMOSTSIMILARLABELS			•
VERBPREPS	•	•	
VERBDISTANCE	•		
DEPDEPTHDIFFERENCE	•	•	
CONDEPTHDIFFERENCE	•		
DEPPATHTOVERBLENGTH	•		
CONPATHTOVERBLENGTH		•	

The two tasks of argument identification and syntactic classification share several features: more precisely, most of the features that are useful for argument identification are also useful in classification while this is not true the other way around. A quick look at the section containing VERB\* features, shows that most of these are only used in classification. This is expected, since, for example, a combination of features like VERBPOSITION and VERBVOICE is very useful in discriminating between a subject (which usually appears before/after the verb if the sentence is in active/passive form) and an object, which behaves inversely, but not very discriminative for argument identification in general. As could be expected, the VERBLEMMA feature does not improve the performance of any classifier, because the test/dev sets use sentences with different verbs than those seen during training. For this reason, many features derived from word embeddings (e.g., VERBSIMILARWORDS, TOKENSIMILARWORDS) proved to be really useful for semantic classification, allowing the classifier to “understand” when a new verb is similar to one it has already seen, thus better generalizing to new unseen verbs instead of treating them as completely unrelated to what it has seen during training.

Classifier	Precision	Recall	Fscore
Argument identification	0.862	0.751	0.802
Syntactic classification	0.945	—	—
Semantic classification	0.695	—	—

Table 8: The performance of the three models on the test set, with the best subset of features resulting from the application of the feature selection algorithm.

Table 8 shows the performance of the classifiers with the best subsets of features obtained by the feature selection algorithm. Recall and f-score are only shown for argument identification, because, as previously mentioned, recall of syntactic and semantic classification is a direct result of that of argument identification and is not considered during the feature selection process since the three steps are optimized independently from each other.

## 4 Results

The performance of the system has been compared to the three other systems that participated at SemEval 2015 task 15 and to that of the baseline provided by the organizers of the task. The following results are obtained using the best performing features for each classifier as detailed in Table 8. The performance of each system, including mine and the baseline, is shown in Table 9.

System	F-score
My system	<b>0.661</b>
baseline	0.624
FANTASY	0.589
BLCUNLP	0.530
CMILLS	0.516

Table 9: Comparison of the scores of the CPA parsing systems.

As previously noted, my system improves on the baseline with an increment of almost 4% f-score and on FANTASY, the best performing system submitted by participants, with an increment of around 7%. This final score, as explained in Section 3.2, is calculated as the average f-score for all the verbs in the test set, so a detailed breakdown of precision, recall and f-score for syntactic and semantic classification for each of them is shown in Table 10.

Verb	Syntactic stats			Semantic stats			Average stats		
	precision	recall	fscore	precision	recall	fscore	precision	recall	fscore
crush	0.836	0.729	0.779	0.484	0.436	0.459	0.657	0.582	0.617
continue	0.92	0.84	0.878	0.634	0.578	0.605	0.777	0.709	0.741
operate	0.788	0.532	0.635	0.327	0.21	0.256	0.563	0.371	0.447
decline	0.898	0.838	0.867	0.626	0.578	0.601	0.763	0.708	0.734
undertake	0.754	0.717	0.735	0.585	0.546	0.565	0.67	0.632	0.65
apprehend	0.825	0.722	0.77	0.745	0.634	0.685	0.786	0.678	0.728
appreciate	0.908	0.713	0.798	0.716	0.559	0.628	0.812	0.636	0.713
<b>AVERAGE</b>	<b>0.847</b>	<b>0.727</b>	<b>0.78</b>	<b>0.588</b>	<b>0.506</b>	<b>0.543</b>	<b>0.718</b>	<b>0.617</b>	<b>0.661</b>

Table 10: Detailed scores for each verb in the test set, obtained by using the scorer for the SemEval task available on the official website

Syntactic classification is definitely the best performing part of the system and analyzing in detail the contribution of precision and recall to the final syntactic score, it can be seen that precision is much higher, which is usually a desirable property in a system that aims at minimizing the work that human annotators have to do. Semantic classification has much lower scores, both in terms of precision and recall. As explained in Section 3, semantic classes in the training set follow a very skewed distribution with common classes such as *Human* appearing for 30% of the tokens. To show how the frequency of a semantic class impacts the f-score that the classifier ends up having, a plot of

the f-score of each class with respect to its frequency in the training set is shown in Appendix B. It is evident that there is a strong correlation between the number of examples and the f-score for each semantic class and, indeed, the best performing classes are the most frequent in the training set.

Category	#Gold	CMILLS	FANTASY	BLCUNLP	baseline	My system
subj	1,008	0.564	0.694	0.739	<b>0.815</b>	0.785
obj	777	0.659	0.792	0.777	0.783	<b>0.817</b>
Human	580	0.593	<b>0.770</b>	0.691	0.724	0.726
Activity	438	0.450	0.479	0.393	0.408	<b>0.571</b>
acomp	308	0.545	0.418	0.702	<b>0.729</b>	0.705
LexicalItem	303	0.668	<b>0.830</b>	0.771	0.811	0.766
advprep	289	0.621	0.517	0.736	<b>0.845</b>	0.817
State Of Affairs	192	0.410	0.276	0.373	0.211	<b>0.529</b>
Institution	182	0.441	<b>0.531</b>	0.483	0.461	0.512
Action	115	0.421	<b>0.594</b>	0.526	0.506	0.372

Table 11: F-scores for the top 10 most frequent classes, compared to those obtained by the other systems.

Table 11 shows the performance of the systems on the top 10 most frequent classes, not distinguishing between syntactic and semantic ones. It can be seen that the baseline is still strong at predicting syntactic arguments using a rule-based approach, but gets beaten in every semantic class. The FANTASY system, which was the best performing one in this task, is not accompanied by an article, so it is not possible to analyze their results in depth.

## 5 Conclusions

The CPA parsing task, which has been discussed in this report, is the first step in the CPA annotation process and consists in identifying and classifying, from a syntactic and semantic perspective, the relevant arguments of a target verb in a sentence so that patterns can subsequently be extracted for each verb and clustered together according to their similarity. Syntactic classification had to be performed on a small syntactic tagset comprising the most common syntactic functions (subject, object, indirect object, adjective complement, adverbial complement) while semantic classification used the CPA ontology, which contains around 250 hierarchically organized semantic classes.

CPA parsing proved to be a difficult task, as none of the systems that participated at SemEval 2015 Task 15 managed to beat the rule-based baseline that was provided by the task organizers, with the best performance being very close but still 3.5% lower than that of the baseline. The approach described in this report outperforms these systems and slightly improves on the baselines results as well, with an increase of almost 4% in terms of f-score. The system uses a pipeline of three maximum entropy classifiers, which, in turn, first identify the relevant verb arguments and then classify them syntactically and semantically.

Among these three steps the argument identification one is arguably the most important in the whole process, as errors at this stage will propagate to all the following steps and its performance is an upper bound to that of the whole system. In the proposed system, this stage performs well, with an f-score of 80%. Syntactic classification, however, is the best performing step in the pipeline: this is probably due to the fact that, given the way the data is structured, the training set contains a high and balanced number of examples for each syntactic class, resulting in easier training. While performance on the syntactic level can be considered good, that of the semantic layer is, unfortunately, very far from those of both argument identification and syntactic classification. There are multiple reasons for this to happen, but, most likely, this is due to the data sparsity problem that characterizes semantic classes in the training set and that has been previously discussed in Section 3.1.

These three classifiers are trained with several kind of features, some inspired from previous work on Semantic Role Labeling and CPA parsing, while others are novel features or variations on known features designed by me during my work on this task. Given the high number of features, manually testing the effectiveness of each one is a hard task, so a simple feature selection algorithm has been employed in order to select the best subset for each of the three models. Despite this, running the feature selection algorithm is still computationally expensive and I have been limited in the number of iterations I could actually execute. I believe that better feature subsets can be found and also that experimenting with other feature selection techniques could lead to improvements in the performance of the system.

In conclusion, I presented a learning-based approach to CPA parsing which showed to perform better than previous attempts, thanks to descriptive features tailored to each of the subtasks that CPA parsing is made of. While most of previous CPA parsing systems either used rule-based approaches or relatively simple features for semantic classification (for example named entities) this work shows how dependency parse trees and WordNet and word embeddings can be used to derive a useful set of features to improve the performance of syntactic and semantic classification respectively.

## References

- [1] Mills, Chad, and Gina-Anne Levow. "CMILLS: Adapting Semantic Role Labeling Features to Dependency Parsing."
- [2] Manning, Christopher, and Dan Klein. "Optimization, maxent models, and conditional estimation without magic." Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Tutorials-Volume 5. Association for Computational Linguistics, 2003.

- [3] Manning, Christopher D., et al. "The Stanford CoreNLP Natural Language Processing Toolkit." *ACL (System Demonstrations)*. 2014.
- [4] Kolb, Peter. "Disco: A multilingual database of distributionally similar words." *Proceedings of KONVENS-2008, Berlin* (2008).
- [5] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
- [6] Feng, Yukun, Qiao Deng, and Dong Yu. "BLCUNLP: Corpus Pattern Analysis for Verbs Based on Dependency Chain." *Proceedings of SemEval* (2015).

# Appendices

## A List of features

1. `TOKENLEMMA` The lemma of the token.
2. `TOKENWORD` The word form of the token.
3. `TOKENPOS` The POS tag of the token.
4. `LEMMASAROUNDTOKEN` Lemmas of the tokens immediately before and after the current one. If the token is at the start or at the end of the sentence, special `<s>` or `</s>` symbols are used to represent the previous or next token respectively.
5. `WORDSAROUNDTOKEN` Words immediately before and after the current token. If the token is at the start or at the end of the sentence, special `<s>` or `</s>` symbols are used to represent the previous or next token respectively.
6. `POSAROUNDTOKEN` POS tags of the tokens immediately before and after the current one. If the token is at the start or at the end of the sentence, special `<s>` or `</s>` symbols are used to represent the previous or next token respectively.
7. `TOKENISVERB` Whether the token is a verb. A token is considered a verb if its POS tag starts with `V`.
8. `TOKENISPREPOSITIONOFVERB` Whether the token is a prepositional dependency of the target verb, that is whether the target verb node in the dependency parse tree has a *prep* dependency relation with the token as child. To compute this feature CoreNLP basic dependencies annotations are used (as opposed to the collapsed dependencies annotations that are used to compute the other features), because when using the other types of dependencies prepositions are collapsed into the edges and they don't have a corresponding node in the tree.
9. `TOKENPHRASETYPE` The phrase type the token belongs to according to the constituency parse tree. Formally, the value of the first non pre-terminal ancestor of the token node (the pre-terminal node contains POS tag annotations); in the case of the "Universities" token the value is `token_phrase_type=NP`.
10. `TOKENPHRASESTRUCTURE` This feature represents the structure of the subtree the token belongs to. As for the `TOKENPHRASETYPE` feature, the first non pre-terminal ancestor of the token node is identified, and then all its children are listed in left to right order; as for the example token the value is `token_phrase_structure=NP->NNS`.



11. `TOKENISSUBJOROBJ` Whether the token has a parent with a *nsubj*, *nsubjpass* or *dobj* relation in the dependency parse tree.
12. `TOKENISVERBCHILD` Whether the token is a child of the target verb in the dependency parse tree.
13. `TOKENISCAPITALIZED` Whether the token is capitalized. This feature should be most useful for semantic classification, in recognizing many proper names that appear in the dataset.
14. `TOKENCONTAINSDIGIT` Whether the token contains digits. This feature should be most useful for semantic classification, in recognizing many instances of the *Numerical Value* semantic class.
15. `TOKENISUPPERCASE` Whether the token is completely uppercase. This feature should be most useful for semantic classification, since, for example, many instances of the *Business Enterprise* class are written in all uppercase characters.
16. `TOKENRELFROMVERB` The direct relation in the dependency parse tree from the target verb to token (if any).
17. `TOKENISUNIQUESUBJOROBJ` Whether or not the current token is the only one in the sentence to have a parent with a *nsubj*, *nsubjpass* and *dobj* relation. More specifically, for each of these relations *R* the output is a boolean feature (so this actually adds three features at most) that takes its value according to the following rules:
  - if the token doesn't have a parent with relation *R*, this feature is not added;
  - if the token has a parent with relation *R* but there exists, in the dependency parse tree, two other tokens linked by *R*, the feature is set to false;
  - otherwise it is set to true.

As for the example, the features for the *nsubjpass* and *dobj* dependencies will not be set, since the "Universities" token doesn't have such relations, and the feature for *nsubj* will be set to `is_unique_nsubj=true` because it is the only token in the whole dependency parse tree to have a parent linked by the *nsubj* relation.

Features computed on the target verb:

18. `VERBLEMMA` Lemma of the target verb.
19. `VERBPOS` POS tag of the target verb.
20. `LEMMASAROUNDVERB` Lemmas of the tokens immediately before and after the target verb. If the target verb is at the start or at the end of the sentence, special `<s>` or `</s>` symbols are used to represent the previous or next token respectively.

21. POSAROUNDVERB POS tags of the tokens immediately before and after the verb. If the verb is at the start or at the end of the sentence, special <s> or </s> symbols are used to represent the previous or next token respectively.
22. VERBVOICE Whether the target verb has active or passive voice, determined through the use of dependencies: if the verb has any child with a relation of *nsubjpass*, *csubjpass*, *auxpass* or *agent* the voice is set to passive, otherwise it is set to active.
23. VERBPOSITION Whether the current token is before or after the target verb.
24. ISVERBPREPOSITIONAL Whether the target verb belongs to a list of prepositional verbs. The list contains verbs that commonly take prepositional arguments like “continue” (*continue to*), “thank” (*thank for*), etc... and it is shown in Appendix A.
25. VERBBY Whether the target verb has a *prep\_by* dependency. If this happens, the dependant is probably a passive subject.
26. VERBPHRASESTRUCTURE The same as the ARGUMENTPHRASESTRUCTURE feature, but computed for the target verb token.  
In the example: `verb_phrase_structure=VP->VBD-S`
27. VERBISROOT Whether the target verb is the root of the sentence in the dependency parse tree.
28. VERBHASNsubj Whether the target verb has a *nsubj* dependency.
29. VERBHASNsubjpass Whether the target verb has a *nsubjpass* dependency.
30. VERBHASDOBJ Whether the target verb has a *doobj* dependency.
31. VERBHASIOBJ Whether the target verb has a *iobj* dependency.
32. VERBHASCcomp Whether the target verb has a *ccomp* dependency.
33. VERBHASAcOMP Whether the target verb has a *acomp* dependency.
34. VERBHASXCOMP Whether the target verb has a *xcomp* dependency.
35. VERBPARENTLEMMA Lemma of the parent of the target verb in the dependency parse tree (if any).
36. VERBPARENTPOS The POS tag of the parent of the target verb in the dependency parse tree.
37. VERBFIRSTVpPARENTLEMMA Lemma of the first ancestor of the target verb that is also a verb: if the parent of the target verb is a verb then this feature is the same as VERBPARENTLEMMA.
38. RELVERBPARENTTOVERB The relation from the target verb parent to the target verb in the dependency parse tree (if any).

The following features are based on paths between nodes in the dependency parse tree. Six types of paths are considered and each path can be augmented with lemmas or POS tags of the traversed nodes, giving rise to a total of 18 feature types based on dependency paths.

39. TOKENDIRPATHFROMVERB

40. TOKENDIRPATHFROMVERBWITHLEMMA

41. TOKENDIRPATHFROMVERBWITHPOS

The shortest directed path in the dependency parse tree from the target verb to the current token. The path is the list of dependency relations that are traversed to reach the candidate token from the verb and it can be augmented with lemmas or POS tags of the traversed nodes. In the example sentence shown before these three features will have the following values:

- token\_dir\_dpath\_from\_verb=nsubj
- token\_dir\_dpath\_from\_verb\_with\_lemma=nsubj-continue
- token\_dir\_dpath\_from\_verb\_with\_pos=nsubj-VBD

If such a path does not exist this feature is not set.

42. TOKENUNDDPATHFROMVERB

43. TOKENUNDDPATHFROMVERBWITHLEMMA

44. TOKENUNDDPATHFROMVERBWITHPOS

The shortest undirected path between the target verb and the current token in the dependency parse tree, augmented with lemmas and POS tags of the traversed nodes similarly to the previous three features. This path is guaranteed to exist, so this feature is always set.

45. DIRDPATHVERBVPARENTTOVERB

46. DIRDPATHVERBVPARENTTOVERB

47. DIRDPATHVERBVPARENTTOVERB

The shortest directed path from the first VP ancestor of the target verb to the target verb.

48. TOKENDIRDPATHFROMVERBVPARENT

49. TOKENDIRDPATHFROMVERBVPARENTWITHLEMMA

50. TOKENDIRDPATHFROMVERBVPARENTWITHPOS

The shortest directed path from the first VP ancestor of the target verb to the current token.

51. TOKENPARENTDIRDPATHFROMVERB

52. TOKENPARENTDIRDPATHFROMVERBWITHLEMMA

53. TOKENPARENTDIRDPATHFROMVERBWITHPOS

The shortest directed path from the target verb to the token parent.

- 54. `TOKENDIRDPATHFROMVERBPARENT`
- 55. `TOKENDIRDPATHFROMVERBPARENTWITHLEMMA`
- 56. `TOKENDIRDPATHFROMVERBPARENTWITHPOS`  
The shortest directed path from the verb parent in the dependency parse tree to the current token.

Two features based on paths in the constituency parse tree:

- 57. `TOKENCPATHFROMVERB` The path in the constituency parse tree from the target verb to the current token.
- 58. `TOKENCPATHFROMVERBPARENT` The path in the constituency parse tree from the parent of the target verb to the current token.

Purely semantic features:

- 59. `HYPERNYMSMCS` All the inherited hypernyms of the current token, obtained from WordNet, up to the “entity” node (excluded). The lemma of the token and its POS tag are used to perform a WordNet lookup and the first WordNet sense (that usually is the most common one) is chosen. For each of the hypernym synsets the first word is taken and used as a feature. In the case of the example, four features are added:
  - `hypernyms_mcs=body`,
  - `hypernyms_mcs=social_group`,
  - `hypernyms_mcs=group`,
  - `hypernyms_mcs=abstraction`.
- 60. `HYPERNYMSDISAMBIGUATED` Same as the `HYPERNYMSMCS` feature, but instead of selecting the most common sense makes use of the output of the disambiguation step done during preprocessing. If the current token has a Babel synset set, then the corresponding WordNet synset is obtained and the hypernyms are computed from that synset; if the token doesn’t have a Babel synset the feature is not set. Although BabelNet provides relations between synsets, and thus hypernyms could have been computed without passing through WordNet, the latter has been preferred for the quality of its manually built taxonomy that, compared to that of BabelNet (that is built automatically) is more accurate and contains less noise.
- 61. `VERBHYPERNYMSMCS` Same as `HYPERNYMSMCS` but computed on the target verb.
- 62. `TOKENSIMILARWORDS` Top 50 similar words to the lemma of the current token. The words are obtained using the DISCO Java library [4] and a word2vec [5] model computed on the English Wikipedia.
- 63. `VERBSIMILARWORDS` Same as the previous feature, computed for the verb.

64. **TOKENMOSTSIMILARLABELS** The most similar semantic classes to the current token. Exploiting the compositionality of word2vec vectors, a vector representing each semantic class is computed as the average of the vectors of the (lowercased and lemmatized) words that compose the name of the class. For example, the vector for the *Abstract Entity* class is the average of the vector for “abstract” and “entity”. These vectors are precomputed for each semantic class, and then, given a token, its cosine similarity with every class is computed: the top 10 most similar classes are used as features.

Real-valued features:

65. **VERBPREPS** This feature is an indicator of how likely it is that the target verb takes a prepositional argument and how related the verb is to each preposition. To compute this, a list of all the words tagged *advprep* is gathered from the training and test set (see Appendix A for the complete list of words): most of these words are prepositions but some are not; nonetheless, I will refer to them as prepositions in this section because they are to be considered prepositional arguments for the verb. For each word  $p$  in this list, and each verb  $v$  in the training and test set, the probability that verb  $v$  has an adjacent preposition  $p$  is computed as:

$$P(v + p) = \frac{\text{count}(v + p)}{\text{count}(v)}$$

where  $\text{count}(v + p)$  is the number of times that the verb occurs adjacent to  $p$  and  $\text{count}(v)$  is the occurrences of the verb. These values are computed on a corpus of around 300000 pages taken from the English version of Wikipedia. This feature is similar to the one used in the verb classification module implemented in [6], with some differences. First of all, they compute both the probability that a verb takes a prepositional argument (based on the dependency parse tree) and the probability that the argument is adjacent to the verb: since dependency parsing is slow and requires a lot of time with limited computational power, especially if parsing of a lot of sentences is necessary, I preferred to only compute the probability of the preposition being adjacent to the verb, thus avoiding the need to parse each sentence, since, although this may be less precise, it is very likely that if a preposition is adjacent to a verb it is also one of its dependencies. Secondly, one value is computed for each preposition, instead of computing just one feature grouping all of them: this allows to differentiate verbs that are related to different prepositions from each other. For example, as can be seen from Table 12, that shows the value of this feature for the verb “continue”, this verb is highly correlated with the preposition “to” and not, or slightly, correlated with all the other ones. Computing a value for each preposition allows to know which one of them the verb is more related to, instead of just knowing whether the verb is likely to take an unspecified prepositional argument.

to	in	on	into	through	from	over	how	...
0.451	0.031	0.023	0.011	0.010	0.003	0.002	0.001	...

Table 12: Value of the VERBPREFS feature for the verb “continue”. It can be seen that the verb is highly correlated with the preposition “to” and slightly correlated with all the other ones.

66. VERBDISTANCE The distance between the token and the target verb, measured in number of tokens.
67. DEPDEPTHDIFFERENCE The difference in depth in the dependency parse tree between the token and the target verb.
68. CONDEPTHDIFFERENCE The difference in depth in the constituency parse tree between the token and the target verb.
69. DEPPATHTOVERBLENGTH The length of the shortest directed path between the token and the target verb in the dependency parse tree.
70. CONPATHTOVERBLENGTH The length of the path between the token and the target verb in the constituency parse tree.

## B F-scores with respect to class frequency

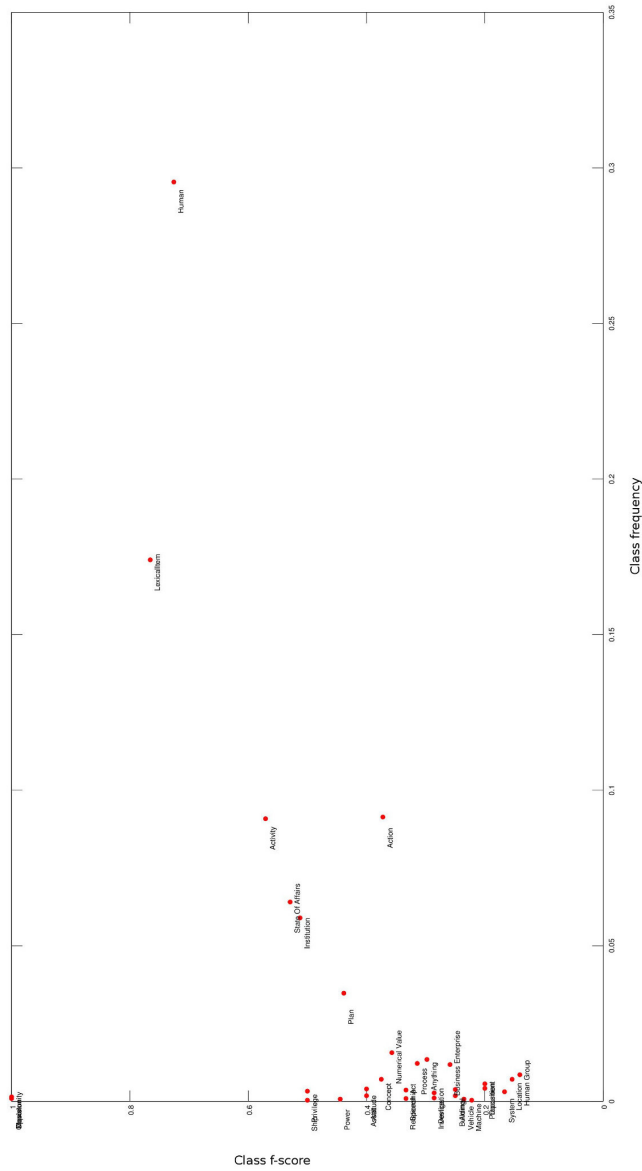


Figure 4: F-scores for semantic classes with respect to the class frequency in the training set. Classes whose f-score is 0 are not plotted.